A Minimization Package for HEP

M. Fischler
D. Sachs, M. Paterno, W. Brown

A Minimization Package for HEP

- Meant to replicate and extend Minuit
  - Same philosophy – ideally suited for HEP fitters
- C++ from the start
- Adherence to accepted OO design considerations
  - Advantages from user standpoint
  - Advantages for internals
- "Stand-alone"
- Focus on being easy to extend and maintain

A Minimization Package for HEP

- Why do this?
- What should the package do and be?
- Concepts that Minuit deals with
- Subsystems in the new package
- New concepts the package will handle
- Some design considerations

Why Do This?

- Physicists have two categories of reactions
  - "It's about time we had this"
  - "Are you out of your mind? Somebody must already have done this!"
- But is a suitable minimization package really available?

C++ minimization via Minuit wrappers

- Root has minimization
  - f2c followed by some hand code cleaning
  - some OO features available (e.g., multiple problems at once)
  - maintainability and extensibility likely to be tough
  - some prefer to avoid linking to such large libraries
- Gemini
  - Wraps either NAG or Minuit
  - Minuit form at least is still tied to the Fortran code

Commercial C++ minimization code

- The HEP community dislikes the paperwork involved in paying money, the licensing issues, and the going-out-of-business risks
- For HEP purposes, Minuit is typically considered superior
  - Evolved specifically to meet HEP needs

Why replicate and extend Minuit?

- Who is maintaining MINUIT?
  - Perhaps Minuit does not need maintenance(?)
  - What happens when CERNLIB Fortran codes start becoming awkward to bring forward? Will Minuit also fall victim?

Extensibility is important

- Minuit hasn't been touched in ~10 years – missing the latest improvements and innovations
  - Algorithms such as linear/quadratic programming and genetic methods may be useful
- A good minimization framework has uses outside the classic HEP parameter fitting problem
- Minuit does have known weaknesses

What Would This Package Have To Be?

- Mimic Minuit's behavior
  - Precisely the same way, if the user so chooses
  - Improvements should not preclude use of the original behavior
- Obtain classic object-oriented benefits
  - independence of sub-systems, and so forth
- User interface must be natural and lead to readable user code

What Would This Package Have To Be?

- Easy to use
  - Good user documentation
  - Limited set of concepts for user to understand
- Easy to maintain
  - Coding must strive for clarity and readability
  - Clear organization of constructs
  - Full documentation of algorithms and coding
- Easy to extend
  - A new good algorithm should not require an expert C++ developer to insert it.

Concepts in Minuit

- Algorithms
- Domain
- Termination criteria
- Solution state
- Solution analyzers
- User function

Algorithms

- Strategies that take some starting point and other information, and move to a "better" point
- Migrad, Simplex, combinations of strategies, …

The Domain

- Minuit doesn't call it "domain", but this corresponds to the notion of restricting the ranges of parameters, and of fixing/releasing values.
- The domain concept is fundamentally a mapping between an "exterior" space that the Function works with, and a simple unlimited Cartesian space that all algorithms can deal with.

Termination criteria

- Algorithm-originated (point of diminishing returns)
  - Migrad won't continue if estimated distance to minimum is less than .001 of its meaningful change scale
- Overall (user criteria)
  - Number of function calls, time spent, estimated accuracy, …

Solution State

- Minuit has COMMON blocks
- We shouldn't

Solution Analyzers

- E.g., Contour

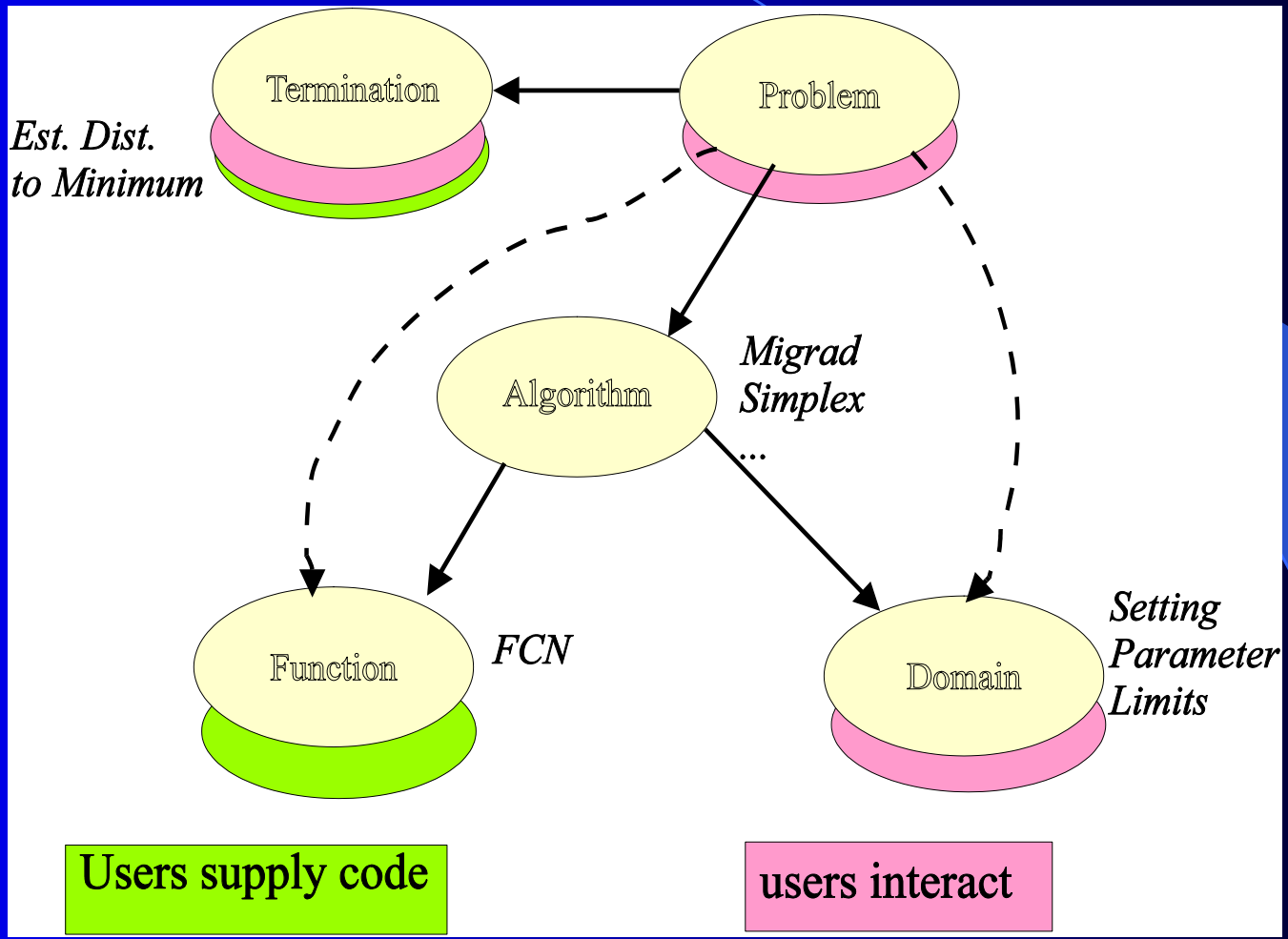
The (user) Function

- Minuit assumes function evaluation is costly
  - Bookkeeping activities are relatively quick
  - This package relies on that notion as well
- There are other possible cases
  - Bookkeeping is expensive (millions of parameters)
  - You need billions of minimizations

Decomposition and subsystems

- Why decompose the package?
  - Extensions are localized
    - E.g., adding a new termination criterion should not involve Domain or Algorithm or ...
  - Simplifies testing
- What is wanted:
  - A well-defined role for each subsystem
  - Minimal subsystem interdependence

Identified Subsystems and Interactions



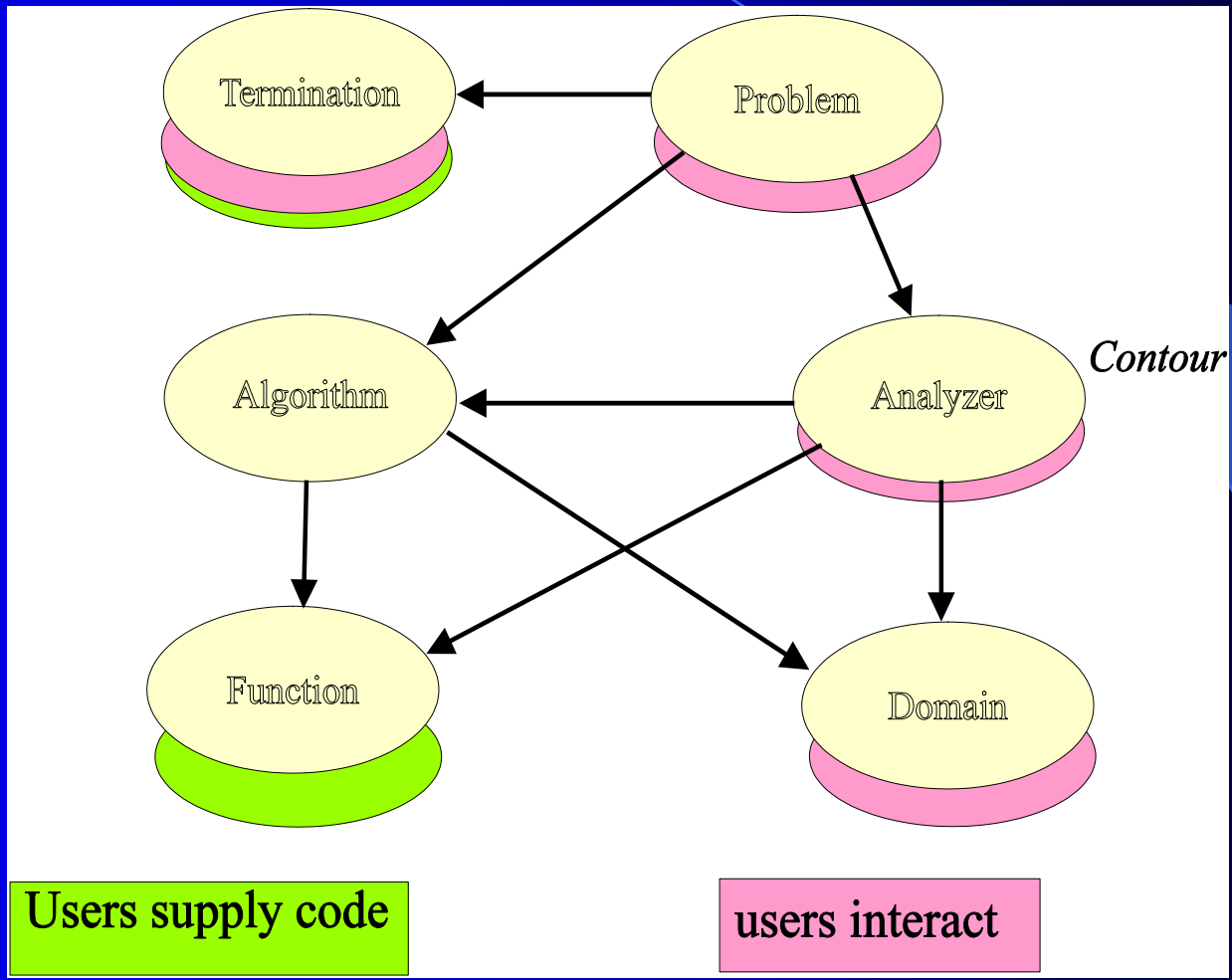
Identified Subsystems and Interactions 1

- *Problem* provides the user interface:
  - for associating functions, domains, termination criteria, and algorithms
  - for controlling the steps taken
- *Algorithm* is responsible for improvement of the best-guess solution

Identified Subsystems and Interactions 2

- *Domain* translates simple coordinates (used by algorithms) into possibly restricted coordinates (understood by a user function), and vice versa
- *Function* subsystem provides the interface to users' functions
- *Termination* provides stopping criteria and means to form compound criteria

Identified Subsystems and Interactions



Sample of Enhanced Concepts

- Generalization of *Domain* concept
  - Minuit supports a particular Domain:
    - Rectilinear (coordinates are separable)
    - A specific style of mapping function
    - Finite or unrestricted ranges per coordinate
  - We plan a more flexible Domain concept:
    - Variety of mapping functions
    - Semi-infinite ranges per coordinate
    - Non-separable coordinates (e.g., spherical)

As close as possible, but no closer

- Because Minuit does something one way, there is a temptation to do it that way without thinking about it
  - In the extreme, this loses all advantages over f2c.
- Requiring a capability is not the same as specifying how that capability is to be achieved

One example pattern

- How we go about making gradient optional?
  - Algorithms that need grad have a way to get it from calls to f ( ) but prefer to use the grad directly if available
- "switch" pattern
  - AL probes f to see if grad is available;
  - if not it uses its own technique
- Better pattern
  - AL requests gradient from Function but supplies the fallback method
  - If the user has overrided gradient ( ) in her concrete Function class AL gets that
  - If not, the gradient ( ) of the base class "calls back" to the supplied method!

Summary

- There is a need for a C++ standalone minimization package in the HEP community and elsewhere
- Needs Minuit's capabilities, and more
- Development of such a package is under way